Taskmonitor User Documentation

Introduction 1
Basic Mechanism 1
Taskmonitor APL Functions 1
Taskmonitor States 2
Task Types 3
Function Details 3
pcdcÒconnect 3
pcdcÒtimeout 4
pcdcÒabort 4
pcdcÒdisconnect 5
Taskmonitor Environments 5
Location 5

Introduction

Taskmonitor, a subset of the Production Control Data Console, is a generalized APL Ntask monitoring facility, which monitors the current state of tasks which have been enrolled in the Console system. In addition to the existence of a task, *Taskmonitor* (*Taskmonitor*) can also be used to ensure a task does not hang indefinitely without being noticed. Tasks which are enrolled with *Taskmonitor* exist in one of eight possible states: Blank, Late, Active, Hung, Done, Crashed, Aborted, or Unclear. How a task moves amongst these eight states is described below.

Information which *Taskmonitor* reports to the Console is displayed as it arrives in the bottom (Logfile) window of the Console. This information is also used to display the current state of the task in the top (Task) window of the Console.

Basic Mechanism

Taskmonitor is an Ntask which runs indefinitely sitting in a Ásc loop, and which will couple shared variables of the correct name and structure/content. Once coupled, the value of a share determines whether or not a task has "timed out" (ie, has become "hung"). If a share is broken (ie, *Taskmonitor* notices that the level of coupling has dropped from 2 to 1), it inspects the last value of the share to see if the uncoupling was planned or not.

Taskmonitor APL Functions

Functions to couple and uncouple with *Taskmonitor* are listed below, and (after enrolling) a task has only to install these functions to work with *Taskmonitor*. No Ásvc's are employed; a task has no fear of hanging when using the *Taskmonitor* functions. Indeed, even if *Taskmonitor* were not running (unlikely though that may be!), a task would carry on without disruption (athough, of course, no monitoring would take place).

A brief outline of the functions follows; a more detailed description appears later in this document.

Function	Description
® pcdcÒconnect ²	Connect as task ² with *Taskmonitor*: establish share, set timeout value to infinity (ie, task will not timeout).
pcdcÒtimeout ²	Reset the timeout value to ² (timeout in ² minutes).
pcdcÒabort ²	Indicate to *Taskmonitor* that task is ending abnormally (² contains information to be displayed on the Console).
pcdcÒdisconnect	Indicate to *Taskmonitor* that task is ending normally.

Taskmonitor States

As mentioned above, there are 8 distinct states that a task can be in from *Taskmonitor's* point of view:

State	Code[1]	Description
Blank[2]	(blank)	Initial state for tasks when the APL system comes up at the beginning of the week (or whenever else it was down).
Late	**LATE**	Task has failed to connect with *Taskmonitor* within 1 hour of the APL system coming up (Permanent tasks only; described later).
Active	**ACTV**	Task is connected to *Taskmonitor*, and has not timed out.
Hung	**HUNG**	Task is connected, but the share value indicates that it has "timed out"; ie, it has not refreshed the share recently.
Done	**DONE**	Task has uncoupled the share after setting it to a value which indicates a normal (nonerror) shutdown.
Aborted	**ABRT**	Task has uncoupled the share after setting it to a value which indicates an error situation (by using pcdcÒabort).
Crashed	**CRSH**	Share has been uncoupled without setting any special value; this indicates that the uncoupling was unplanned; an error.
Unclear	**????**	The task is coupled with *Taskmonitor*, but the share value has been corrupted. Should not happen with normal processing.

State changes for tasks take place as follows:

Get From state	To state	By using
(blank), DONE, LATE, CRSH, ABRT	ACTV	pcdcÒconnect
HUNG, ????	ACTV	pcdcÒtimeout
ACTV, ????	HUNG	(failing to invoke pcdcÒtimeout before timing out)
ACTV, HUNG	????	(corrupting the share)
ACTV, HUNG, ????	DONE	pcdcÒdisconnect
(blank)	LATE	(failing to connect to *Taskmonitor* when APL starts)
ACTV, HUNG, ????	CRSH	(an unplanned uncoupling of the share)
ACTV, HUNG, ????	ABRT	pcdcÒabort

[1] Code which appears in the Task Window of the Data Console.
[2] Note on the Blank state: Tasks which were in the ACTV, HUNG, DONE, LATE and ???? states are moved to Blank when the system comes up; TMon "remembers" any CRSH'ed or ABRT'ed tasks, and leaves them in that state when APL re-starts.

Task Types

There are two basic task types from *Taskmonitor's* point of view: Permanent and Recurring. A Permanent task is a task which is supposed to run for the entire time that APL is up. Such tasks include servers and monitors which have work to do all week long. A Recurring task, on the other hand, is only up for certain times during the week. Such tasks include monitors which are scheduled to run for certain time periods each day (or each business day), or tasks which are started by a regular event happening (incoming transmission, processing spawned by someone doing work on a system, etc).

Permanant tasks must connect with *Taskmonitor* within an hour of the APL system coming up, or they are moved into the error state LATE. There are no such restrictions on Recurring tasks.

Function Details

The four *Taskmonitor* APL functions work as follows:

pcdcÒconnect

The pcdcÒconnect function is used to connect a task with *Taskmonitor*, so that monitoring can begin.

The righthand argument to pcdcÒconnect is a character vector containing the "task name". This task name is a special process name which must be agreed upon with Production Control, and enrolled in *Taskmonitor*. It is a maximum of 12 characters long. Attempting to use pcdcÒconnect with an unknown task name will not impede the task, but, of course, no monitoring will take place.

The lefthand argument to pcdcÒconnect is an optional boolean flag which indicates whether the task wants to connect to *Taskmonitor* or *Taskmonitorx* (see the section "Taskmonitor Environments" below for descriptions). Supplying a 1 indicates that a connection to *Taskmonitor* is desired; a 0 indicates a desire to connect to *Taskmonitorx*. If omitted, pcdcÒconnect defaults the argument to a 1 (ie, connect with *Taskmonitor*).

The pcdcÒconnect function has no result.

The pcdcÒconnect function first ensures that the clone-id of the task is not zero (via Ásvn ¯1). If it IS zero, it sets it to 1↑,Áruns; therefore, tasks which require a specific clone-id should set it BEFORE invoking pcdcÒconnect. Note that it is not necessary to freeze the clone-id (by offering a share) before calling pcdcÒconnect.

After ensuring a reasonable clone-id, pcdcÒconnect initializes the share variable, "pcdcÒsv1". This variable is, of course, global to pcdcÒconnect, and must not be reassigned or erased for the duration of the connection. It can be safely inspected by the connecting task, as no share-locking (Ásvc) takes place. It can be localized in the main "driver" function of the task. The initial value of pcdcÒsv1 indicates the task name (supplied to pcdcÒconnect) and a timeout value of infinity (represented by 0).

The pcdcÒsv1 variable is then offered to *Taskmonitor*, and pcdcÒconnect returns to the calling function (pcdcÒconnect does not wait to see if the share couples, as this could impede the calling task).

pcdcÒtimeout

The pcdcÒtimeout function should be used by a connected task that is about to enter a section of code which may cause it to "hang" indefinitely, and would like the Console to be alerted if such a hang takes place. Events which could cause a task to hang include: performing Árabin, attempting to Áhold a file, and even set/use of a shared variable (if the variable has a Ásvc set and the partner does not use/set). Note that during normal "file backups" the system may hold up any given file op for as long as 15 minutes, although this should occur a maximum of once for any given file.

The righthand (and only) argument to pcdcÒtimeout is a single integer value, which indicates the maximum number of minutes the task could normally be away without being considered hung. If 0 (zero) is passed to pcdcÒtimeout, the timeout value is set to infinity; both '' and ι0 are treated like 0.

Firstly, pcdcÒtimeout checks to ensure that it has been passed a numeric argument and that the pcdcÒsv1 variable exists; if not, its only action is to exit. If these two criteria are met, it reassigns pcdcÒsv1 with the new timeout value. If a task is in the HUNG state, invoking pcdcÒtimeout will return it to the ACTV state.

The pcdcÒtimeout function has no result.

Caution on the use of pcdcÒtimeout: Note that each invocation of pcdcÒtimeout sets the value of the shared variable pcdcÒsv1, which in turn causes a Ásc post to *Taskmonitor*, waking it up. *Taskmonitor* then looks at the value of pcdcÒsv1, which will post the task which set the variable. This can lead to problems if (1) the calling task is in a Ásc controlled loop itself, and (2) it blindly invokes pcdcÒtimeout each time through this Ásc loop. If these conditions apply, an "infinite post" situation will occur, causing both the calling task and *Taskmonitor* to post each other indefinitely, wasting a LOT of cpu cycles and putting a load on the Shared Variable Processor. If the calling task is in a Ásc controlled loop, it should have a conditional invocation of pcdcÒtimeout, ensuring that pcdcÒtimeout is called no more often than, say, once per minute. This can be accomplished by using a "last invoked at time X" variable.

pcdcÒabort

The pcdcÒabort function can be invoked if a task encounters a situation which it knows it cannot recover from, and wants the Console to be alerted of this. Such situations could include Átrap'ed APL errors or a calculation which indicates that further processing is futile.

The righthand (and only) argument to pcdcÒabort is a simple or enclosed vector. If simple, it represents the description field which appears in the Console's Logfile window. If '' (or ι0) is given, the default message will be displayed: either "task aborted (was active)" or "task aborted (was hung)", whichever is appropriate. If an enclosed argument is supplied, it will be a 1 to 3 element vector, with the first element being the description field (as described above). The second element is an optional "zoom" field, which can contain any information which may be of use to the Data Operator (the zoom field can be called up for display on the Console). The third element is an optional "action" field, which can be used to change the default severity of the event (see the Production Control document "DSPCDC User Documentation" for more details).

The pcdcÒabort function assigns pcdcÒsv1 the special abort value, along with the information passed to it, then Áex's the variable, uncoupling the share. The Shared Variable Processor ensures that this last value is delivered to *Taskmonitor*, even though the variable no longer exists in the calling task's workspace.

The pcdcÒabort function has no result.

pcdcÒdisconnect

The pcdcÒdisconnect function is used to indicate a normal, non-error shutdown to *Taskmonitor*. It is niladic.

The function first ensures that pcdcÒsv1 exists; if not, it exits. If the variable does exist, it sets it to the special normal shutdown value, then Áex's the variable, uncoupling the share. The Shared Variable Processor ensures that this last value is delivered to *Taskmonitor*, even though the variable no longer exists in the task's workspace.

The pcdcÒdisconnect function has no result.

Taskmonitor Environments

A total of four *Taskmonitor's* run, two on the Prod system and two on the Intern system. Enrollment is required; contact Production Control:

"*Taskmonitor*" on Prod is the live version, and should be used by tasks which require live monitoring by Data Operators.
"*Taskmonitorx*" on Prod is the test version, and should be used by tasks which are preparing for live monitoring (ie, testing the installing of the functions described above, etc).
"*Taskmonitorx*" on Intern is a test version for systems which have their test setups on Intern.
"*Taskmonitor*" on Intern is the development version. Tasks can attempt to connect to it if they choose, and can do so without risk, but monitoring may be interrupted or erratic, as alpha testing of new code is done on this version.

As mentioned above, the lefthand argument to pcdcÒconnect indicates if a task wants to be connected to *Taskmonitor* or *Taskmonitorx*. In general, a task should not connect with the Intern version of *Taskmonitor*, as new code is released here first for testing, and performance may be interrupted.

The other three functions (pcdcÒtimeout, pcdcÒabort, and pcdcÒdisconnect) do not need a lefthand argument, since the initial connection of the shared variable determines which system is to be used.

Location

All of the above functions are stored in Logos underneath the "pcdc" alias on both Prod and Intern, as shown below (read/execute permission has been granted to Áall) (The dspcdc function is described in the Production Control document "DSPCDC User Documentation"):

```
                    dspcdc
                    pcdcÒabort
.pcdc.src.extfns    pcdcÒconnect
.
                    pcdcÒdisconnect
                    pcdcÒtimeout
```

The above functions are also stored in a workspace on both Prod and Intern: 1650396 pcdcfns

We Console logfile 1650396 pcdclog
test 1650396 pcdclogx